SAGE-26,515

SYSTEM & METHOD OF TABLE BUILDING FOR A PROCESS-BASED SECURITY SYSTEM USING INTRUSION DETECTION

Inventors:

Vincent Alan Larsen
Carolyn Meinel

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

SYSTEM & METHOD OF TABLE BUILDING FOR A PROCESS-BASED SECURITY SYSTEM USING INTRUSION DETECTION

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation-in-part of US patent application 10/061,701 entitled "METHOD AND APPARATUS FOR IMPLEMENTING PROCESS-BASED SECURITY IN A COMPUTER SYSTEM," filed on February 1, 2002.

TECHNICAL FIELD OF THE INVENTION

[0002] The present invention relates in general to software security systems and, more particularly, to a table building for a process-based security system using intrusion detection protocols.

BACKGROUND OF THE INVENTION

[0003] Data processing systems have, as of recent, seen a large increase in the use thereof. For small users, such as home users, a typical system will run multiple programs that will allow access to various stored data files, allow the user to access resources such as disk drives, modems, faxes, etc. Access to these types of systems is typically what is referred to as "unrestricted", i.e., any one possessing the required knowledge to access a given program can access it on any unrestricted computer. However, for a larger data processing system that may contain confidential information, the user may be provided access to resources that are billed on a time-use, etc., these systems usually requiring restricted access.

[0004] In restricted access systems, a user is typically given an I.D. to the system. A system administrator can then configure a system, via a network or even a stand-alone system, to define the user's access to the system, once the user has logged in to the system. For example, in the network environment, there are a plurality of network drives, network resources such as printers, faxes and mailboxes, etc. The user has a configuration file that defines what access the user has. Upon logging in, the network will then access the configuration table and allow that user access to the given system resources. The user can then execute a program and utilize the program to access the resources through something as simple as a disk operating system (DOS). The

disadvantage to this type of access is that the user now has full access to resources for any purpose, other than the purpose for which the user was given access.

[0005] As an example, a user may need access to a modem for the purpose of running database searching software. This database searching software allows the user to dial up a provider with the modem to perform predefined searching. In order for a prior restricted system to utilize the modem, the user must be granted access to a given serial port. However, the user need not run the database searching software in order to have access to the modem. This allows the user to run other programs that can gain access to the system. The disadvantages to this is that, although the database searching software may have certain restrictions that are inherent in the software itself, a user can bypass this system to utilize the modem for other purposes. This can also be the case with respect to data files, wherein a word processing program has the ability to read and write files and gain access to printers through the word processing software. However, this access must be granted in a global manner, such that the user can access the files and printers via any other means, once logged into the system.

[0006] As another example, consider a database that allows access to databases such as payroll, criminal records, etc., which a user has been given access. With current operating system security, the user can certainly go outside of a given program that is utilized with a specific database to copy, delete or even change files in the database outside of the program. As such, there exists a problem in that security for current operating systems provides that resources are allocated based on users or the groups to which the users belong. This therefore allows the user access to those resources even though the process that needs those resources is not being run. These rights will in turn allow the user to use the resource outside of its intended use.

[0007] In a general purpose computer system operating with a wide assortment of applications or processes, usually as part of a bundled package, security is based on the control of user access which allows access to all of the resources on the system whether

they are needed or not. A disadvantage of this system is that it is not very secure and is prone to break-in or misuse of resources, some of which contain very sensitive information, primarily because of the unrestricted access to resources. All that is required to enter such a system is a user ID and a password. One solution, implementing process-based access to a general purpose computer system, where the access to each resource is controlled in addition to the entry of a user ID and password, would provide an additional level of access control.

SUMMARY OF THE INVENTION

[0008] A method to build a resource access table in a system for controlling access to resources is disclosed. The protocol identifies a resource call in a process. Each of the resources accessed by resource calls are identified and analyzed by intrusion detection software. Permission is assigned to the resource, with regard to the process. A resource access table is written, with each entry identifying the process, the resource and the permission. When the process is executed and the process makes a resource call to the resource, access to the resource is controlled by the permission data entry in the resource access table entry of the resource access table.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] For a more complete understanding of the present invention and the advantages thereof, reference is now made to the following description taken in conjunction with the accompanying Drawings in which:

[0010] FIGURE 1 illustrates a block diagram of an overall system;

[0011] FIGURE 2 illustrates a prior art security system;

[0012] FIGURE 3 illustrates a general block diagram of the system of the present invention;

[0013] FIGURE 4 illustrates a more detailed block diagram of the system of the present invention;

[0014] FIGURE 5 illustrates an alternate embodiment of the system of the present invention;

[0015] FIGURE 6 illustrates a flowchart depicting the operation of the present invention;

[0016] FIGURE 7 illustrates a flowchart depicting the resource request by the process;

[0017] FIGURE 8 illustrates a flowchart depicting the processing of the request by the operating system;

[0018] FIGURE 9 illustrates a functional block diagram of an embodiment of a process-based security system according to the present disclosure;

[0019] FIGURE 10 illustrates a flowchart of the operation of the embodiment of the process-based security system of FIGURE 9;

[0020] FIGURE 11 illustrates a flowchart of an example of use of a resource access table according to the embodiment of FIGURE 9;

[0021] FIGURE 12 illustrates a functional block-diagram of a multi-user process based security system according to the disclosure;

[0022] FIGURE 13 illustrates a flowchart of an authentication process used in a process based security system;

[0023] FIGURE 14 illustrates a flowchart of a set password process used in a process based security system;

[0024] FIGURE 15 illustrates a flowchart of a key exchange process used with a process based security system;

[0025] FIGURE 16 illustrates a flowchart of a financial transaction using a process-based security system;

[0026] FIGURE 17 illustrates a flowchart of a secure file transfer process using a process-based security system;

[0027] FIGURE 18 illustrates a flowchart of a self-rebuilding function;

[0028] FIGURE 19 illustrates a flowchart of a table building process in accordance with one embodiment;

[0029] FIGURE 20 illustrates a flowchart of a table building process in accordance with a second embodiment;

[0030] FIGURE 21 illustrates a functional block diagram of an embodiment of a process-based security system using intrusion detection; and

[0031] FIGURE 22 illustrates a flowchart of a table building process using intrusion detection.

DETAILED DESCRIPTION OF THE INVENTION

[0032] Referring now to the drawings, wherein like reference numbers are used to designate like elements throughout the various views, several embodiments of the present invention are further described. The figures are not necessarily drawn to scale, and in some instances the drawings have been exaggerated or simplified for illustrative purposes only. One of ordinary skill in the art will appreciate the many possible applications and variations of the present invention based on the following examples of possible embodiments of the present invention.

[0033] Referring now to FIGURE 1, there is illustrated an overall block diagram of a system for operating in conjunction with the security system of the present invention. A central processing unit (CPU) 10 is provided, which CPU 10 has associated therewith a microprocessor, peripheral circuitry, power supply, etc., all required to execute instructions and, in general, run programs. The CPU 10 has associated external therewith a keyboard 12 to allow the user to input the information thereto and a display 14. A disk input system 16 is also provided which allows the user to download and upload data. The CPU 10 also has a local memory 18 associated therewith, the local memory 18 being in the form of random access memory, read-only memory, flash memory or other forms of memory. The CPU 10 may also include peripheral storage devices including flash memory, optical disk drives such as CD or DVD drives, hard drives, disk drives or any other form of data storage. The local memory 18 is operable to store both files in a region 20 and also execute programs in a region 22. The files in the region 20 can also consist of data for databases. The CPU 10 can then access the executable files in the program portion 22, execute the programs and access information in the form of files and/or data.

[0034] The CPU 10 also interfaces with peripheral resources through an Input/Output (I/O) block 24. The I/O block 24 allows access to such things as a modem 26, a network card 28, a scanner 30 and a printer 32. The scanner 30 and the printer 32 are

referred to as local resources. The modem/fax 26, however, allows access to remote resources, such as a public telephone network (PTN) 34. The network card 28 allows access to a network 36, which in turn allows access to network resources 38.

[0035] Referring now to FIGURE 2, there is illustrated a block diagram of a prior art security system. In prior art security systems, a user represented by a block 40 is allowed to access the system only upon providing the correct user I.D. A system administrator 42 is operable to pre-store configurations for the users in a user resource access template 44. This user resource access template is accessed whenever a user is brought into the system and attempts to log in to the system. Once the user logs in, each user is given a predetermined configuration, this referred to as user access blocks 46, there being one associated with each user. Each of these user access blocks 46 contain the configuration that was pre-stored by the system administrator 42 in the user resource access template. Once configured, this defines how the user is interfaced with the system, the system comprised of a plurality of system resources, represented by blocks 48. In the example illustrated in FIGURE 2, there are provided five system resource blocks 48.

[0036] It can be seen in the prior art system of FIGURE 2 that there are three user access blocks 46, representing three different user configurations, although there could be more, and five separate system resources, although there could also be more of these. The user access block associated with the user No. 1 is associated with system resource 1 and system resource 2. User access block 46 associated with user No. 2 has access to system resource No. 1, system resource No. 3 and system resource No. 4. Associated user access block 46 provides a configuration that allows user No. 3 access to system resource 1 through 5. It is noted that this is independent of the process upon which the user is working. The system resource is accessible by the user and not by the process.

[0037] Referring now to FIGURE 3, there is illustrated an overall block diagram of the present invention. In the present invention, security is based upon a process. A process, such as a word processing program, a financial program, etc., each have certain needs related to the resources. For example, a payroll program, once accessed, would need to have access to its database. The program, however, typically does not allow as an inherent part thereof for the database to be copied, does not allow the database to be deleted nor manipulated in any manner other than that provided by the program itself. These may even have audit trails that cannot be bypassed. The present invention provides for security wherein the resource is only accessible through the step of executing and running the program. If the program is not running, the user does not have access to the given resource unless through another process which happens to have access to the resource.

[0038] Referring further to FIGURE 3, a general operating system 60 is illustrated, this operating system being any type of operating system such as a Microsoft Windows based system, a UNIX system or even a DOS system. An operating system is the primary method by which a computer interfaces between the peripheral systems, such as memory storage devices, printers, modems and a process running on said computer. The user is then provided with a platform on which to run programs wherein the user can access the program and have the program access various peripherals through its interaction with the operating system. Operating system 60 therefore provides the necessary commands to access various resources 62 on the system. Again, these resources can be such things as a modem, a printer, a scanner, etc., even including a magnetic disk drive. The operating system is restricted to allocate only those resources defined in a resource access table 64, which resource access table defines resources associated with a given process, which association is based upon the process' needs.

[0039] The process expresses its need via a process requesting mechanism 66 which is an inherent aspect of process execution for any given process. The process requesting mechanism 66 initiates a request for a resource and, if the process running

on the system has access to that resource, as noted in the resource access table 64, the operating system 60 will then grant the resource for use by the process within the operating system. If not, then the operating system will block access.

[0040] Referring now to FIGURE 4, there is illustrated a more detailed block diagram of the process-based access system of the present invention. In general, a user block 68 is provided which indicates a user of the system. The user of the system can access any given process, being illustrated for processes in process blocks 70. Each of the process blocks 70 is connected to a process access selector 74, each of which is associated with one of the resource blocks 62, there being illustrated five resource blocks 62. The process access selector 74 is operable to potentially allow any process to address that process access selector 74. An access control block 76 is provided that receives as inputs an indication of which of the processes in process block 70 is running. A System Administrator block 75 is provided to allow a System Administrator to set the parameters of the access control block. The access control block then selects which of the process access selector blocks 74 is authorized to have access to a given resource. It is important to note that it is a request from a process block 70 during the running of that process that is utilized by the access control block 76 to grant access to the process access selector 74.

[0041] By way of example, if a word processing program were being operated and, on the same computer, a user had the ability to operate an accounting program, the word processor would be provided access to certain regions on a disk and the files associated therewith. The user could retrieve these files, delete them, modify them and restore them. However, the user would not be allowed through the word processor to access the accounting database for any purpose, since the process does not require this. In another example, if a modem were provided, this would not usually be a resource that was available to a word processor. The modem would, for example, only be accessible to a communications program.

[0042] In another example of the operation of the process based security system, where resources are permitted access only in association with the particular process that is selected, reference is made to Table 1, which is an example of the resource access table 64 of FIGURE 3:

TABLE 1

Step	Process name	Resource name	Rights Mask
1	C:*.*/S	C:*.*/S E:\LOGIN\LOGIN.EXE	Full access Execute only
2	E:\LOGIN\LOGIN.EXE	E:\SYSTEM\PASSWORDS E:\PROGRAMS\MENU\MENU.EXE	Read only Execute only
3	E;\PROGRAMS\MENU\MENU.EXE	E:\PROGRAMS\MENU\SCREENS E:\PROGRAMS*.EXE/S E:\PROGRAMS*.COM/S	Read only Execute only Execute only
4	E:\PROGRAMS\WP51\WP.EXE	F:\LIBRARY*.* G:\COMMON*.*/S	Full access Full access

The example in Table 1 illustrates a general personal computer of the clone type, running an MS DOS operating system which is attached to a network with a process based security. When the computer is started, any process with the C:\ drive (denoted with the wild card processing of *.*) and its sub-directories (denoted with the /S option on the end of the process name) is provided full access to anything on the C:\ drive (once again denoted with the wild card resource name of *.*) and its sub-directories (once again denoted with the /S option on the end of the resource name). The user can also execute the process E:\LOGIN\LOGIN.EXE from the network. All other resources from the network are not available to the computer at this time. This situation represents a user, on a computer, who can log into a network, but has not done so. In essence, the user can do anything with their local resources, but nothing with network resources, until they are identified to the network with the login program.

[0043] In step 2 in Table 1, when the user executes the E:\LOGIN\LOGIN.EXE process, the process changes from something on C:\ to LOGIN.EXE which can read the E:\SYSTEM\PASSWORDS file and execute the E:\PROGRAMS\MENU\MENU.EXE program. The file LOGIN.EXE is the network's method of identifying users of the

network. Execution of LOGIN.EXE will verify the user through its read-only access to the E:\SYSTEM\PASSWORDS file. If the user is verified as a valid user, LOGIN.EXE will pass control on to step 3 and the process E:\PROGRAMS\MENU\MENU.EXE.

[0044] In step 3, when MENU.EXE gets executed, it will read the appropriate menu options from its SCREENS file and display it for the user. MENU.EXE controls what programs can be executed and as such, it has been given rights to execute any program in the E:\PROGRAMS directory or any of E:\PROGRAMS sub-directories (this is denoted with the /S option after the partial wild card name *.EXE and *.COM). In step 4, in the event the user executes the WP.EXE program, this process has full access to a local F:\LIBRARY directory, a shared G:\COMMON directory and the sub-directories of G:\COMMON. The example in step 4 may also represent a network, where personal files are stored in a user-related directory (F:\LIBRARY) and company shared documents are stored in a common directory (G:\COMMON).

[0045] In the preceding example, it can be seen that the user cannot, for example, obtain access to the PASSWORDS file by any other process except for the LOGIN.EXE process and this process determines how the user can deal with that particular file.

[0046] Referring now to FIGURE 5, there is illustrated an alternate embodiment of FIGURE 4, illustrating how a particular process can constitute a resource. The resource blocks include a resource block 80 which constitutes a sixth resource, in addition to the five resource blocks 62. However, this resource also has a process block 82 disposed therein, which constitutes a fifth process. Access to this process block 82 is facilitated through the use of a process access block 84, similar to the process access block 74, and controlled by access control 76. Each of the processes in process blocks 70 have access to the process block 82 and the resource block 80 through the process access block 84. Therefore, if one of the processes in process blocks 70 were running and the access control block 76 allowed access through a process access block 84, then

process No.5 in resource block 80 could be run. This, of course, would then allow process block 82 and the process #5 associated therewith to request and receive access to any of the resources in resource blocks 62 associated with process access block 74 in accordance with the access control information in access control block 76. Although illustrated as only a single process that is accessed by another process, there could be many processes deep, such that three or four processes would need to be run before a given process was accessible which, in turn, could access an end resource.

[0047] It is important to note that the process must be running and, during the running thereof, execute instructions that request a given resource. It is the running of the process and the request by that running process that allows access to a resource. The fact that the process has been opened and initiated is not sufficient to gain access to the resource since it is the use of the resource by a given process that is selected. For example, if a multi-tasking operating system were utilized and a given program executed from that platform as a first task, it may be desirable to place that task in the background and initiate a second task. Even if the first task were running in the background, the ability of the first task to request a given resource does not in any way effect the rights of the second task to have access to that same resource. Unless it is in the resource access table, no access to the resource will be allowed. Even if the first task were operating and it were utilized to "launch" a second process, this would not effect the resource access of the launched process, since when the launched process is running, the launching process is not running and it is not the launching process that is requesting access to the resource. Therefore, it is only the requesting resource that is of concern.

[0048] Referring now to FIGURE 6, there is illustrated a flowchart depicting the overall operation of the system. The program is initiated at a start block 100 and proceeds to a decision block 102. The decision block 102 determines if a process has been initiated. If not, the program flows back around an "N" path to the input of decision block 102. When a process has been initiated, the program will flow to a

function block 104 to log in the user. The log in procedure is a procedure that may be utilized, but can be optional. In and of itself, as described above with reference to Table 1, the log in process is a separate process in and of itself. However, some programs by themselves, require log in procedures, i.e., accounting systems. Therefore, this is an optional block.

[0049] After the log in block 104, the program will flow to a function block 106 to run the process. Once the process is running, the program then flows to a decision block 108 to determine if a resource request has been made by the running process. If not, the program will flow along the "N" path back to the input of function block 106. When a resource request has been made by the running process, the program will flow from decision block 108 along a "Y" path to a function block 110, wherein the resource access table is accessed to determine access rights for the requesting process. The program will then flow to a decision block 112 to determine if access has been granted for that particular resource. If not, the program will flow along a "N" path to a function block 114 to return an "access denied" error code. The program will then flow back to the input of function block 106. However, if access rights have been granted in accordance with the resource access table, the program will flow along a "Y" path from decision block 112 to a function block 116 to allow access to the system and then back to the input of function block 106. This will continue unless the resource is halted.

[0050] Referring now to FIGURE 7, there is illustrated a flowchart depicting the resource request by the process, as in step 108 of FIGURE 6. The flowchart is initiated at a block 120 and then proceeds to a decision block 122, wherein the process determines if a resource is needed for the process. If not, the program flows along an "N" path back to the input of decision block 122. If a resource is required, the program will flow along a "Y" path to a function block 124. The function block 124 then determines the ID for the resource and then generates the request, this request defining the resource that is required and also the mode of access that is required. The program will then flow to a decision block 126 to determine if the resource is available. If not,

the program will flow along an "N" path back to the input of decision block 122. If it is available, the program will flow along a "Y" path to a function block 128 to process the resource in accordance with the operation of the process and then flow to a return block 130.

[0051] Referring now to FIGURE 8, there is illustrated a flowchart depicting the operation of the operating system when processing a request for a resource. This is initiated at a block 134 and then proceeds to a decision block 136. The decision block 136 determines whether a resource request has been received from a process operating in conjunction with the operating system. If not, the program will flow along an "N" path back to the input of decision block 136. If a resource request has been received, the program will flow along the "Y" path to a function block 138. Function block 138 fetches the information stored in the resource access table to determine if the particular resource has any access rights associated therewith. The program will then flow to a decision block 140 to determine if access rights are present in the resource access table for the requested resource. If not, the program will flow along the "N" path to a function block 142 wherein a "NULL" signal is sent back to the requesting process to deny access and then to the input to decision block 136.

[0052] If access rights exist in the resource access table for the given resource, the program will then flow to function block 144 to determine if the mode of access that is requested by the requesting process is present in the resource access table, i.e., whether the resource access table has been set up to grant this mode of access to the given resource. An example of this would be a file that is defined as the resource with the modes being READ and WRITE, with either reading of the file from the disk or writing of the file to disk. The program will then flow to a decision block 146 to determine if the mode of access is available. If not, the program will flow along the "N" path back to the function block 142 and, if the mode of access is available, the program will flow along the "Y" path to a decision block 148 to determine if the requested resource and mode of access are valid for the requesting process. For example, a process may request access to a particular memory device or portion of a memory device and it may require access to that memory device for writing of information thereto. The system

will first determine if the resource has access rights associated therewith and then check to see what mode of access is allowed. In this example, if the resource is available, it may have a mode of access available for reading and a mode of access available for writing. However, the resource access table may only grant reading rights to the requesting process. In this event, access will be denied. This is represented by the path that flows along the "N" path back to function block 142. If access is allowed, the program will flow along a "Y" path from decision block 148 to a function block 150 to grant access and then to a return block 152. The following is the process flow for the process generating the request:

Process

```
id = fopen ("filename", "rt");          -This is the request to
                                         the operating
                                         system for the file
                                         access (resource).

if (id == NULL)
        {
        file not available
        }
else    {
        process the opened file
        }
```

The following is the process flow for the operating system when servicing the request:

Operating System

```
FILE *fopen (char *name, char *mode)
        {
                (before checking for the presence of the file,
                check to see if the process has any rights to the file.)

for (I = ⌀; i<SIZE_OF_ACCESS_TABLE; i + +)
    if (check (name, accesstable [i]. resource) == ⌀)
        break;
if (i == SIZE_OF_ACCESS_TABLE) return NULL; (no rights at all)
for (j = ⌀; j< accesstable [i]. rights_size; j ++)
    if (check (mode, accesstable [i]. rights [j]. Mode) == ⌀)
        break;
if (j == accesstable [i] rights size) return NULL; (specific right not
present)
        (the remaining code deals with what the operating
        system needs to do to allocate the file to the calling
        process (note additional errors may still occur, like
        file not found)).
        }
```

[0053] The foregoing describes a system for providing process-based security to an operating system. In this security system, access to any resource on the system, although provided by the operating system, is restricted by the operating system to only those requested by a given process. Whenever a process, during operation thereof with the operating system, requires or requests access to a resource, the operating system makes a determination as to whether a resource can be made available for that given process. This is facilitated by storing access rights for any process in a resource access table. When a process is running on the system and it requests for its use a given resource, the operating system will then look up in the resource access table to determine if the resource is available and, if so, grant access to the resource.

[0054] It has been described hereinabove that a process-based security system controls access to resources via the process (i.e., application) that requests the resource. Experimentation and development have shown, however, that such a security system is

is particularly efficient when implemented in a dedicated, or single-purpose environment, such as a web server, or other, computer appliance-type of application.

[0055] Thus, it can be appreciated that the process-based security described hereinbelow, as applied to a dedicated, single purpose computer system, could exhibit the following advantages:

(1) prevent a user from loading their own applications into the system;

(2) prevent a user from attempting to access files from uncontrolled processes, e.g., trying to load in a hex editor to obtain access to the sensitive files such as accounting records, etc.;

(3) prevent access to all the resources in a system, even though individual resources are needed by a particular program or process;

(4) process-based security, in a dedicated environment, is self contained, i.e., independent of the rest of the system, and is thus relatively easy to implement in existing systems;

(5) process-based security is context-specific or can be made to be dynamic by the way in which resource access is interpreted; and

(6) in a process-based security system, the user only has the right to execute a specific application or process and that process includes access rights only to specific resources keyed to the requesting process.

[0056] Referring to FIGURE 9 there is illustrated a functional block diagram of an illustrative embodiment of a process-based security system according to the present disclosure. A portion of the functional aspects of a complete computer system 160 is shown having an operating system 162 coupled with a plurality of resources indicated by the three blocks identified by reference number 172, respectively resource A, resource B and resource C. In general, resources 172 may be various types of input and/or output devices or application programs installed on a particular system. I/O device resources may include a keyboard, mouse, scanner, display, modem, disk drive,

printer, or an interface to a network, etc. Application, or process type resources may further include a word processor, a spreadsheet, a communication or e-mail program, a database, a search engine or browser and the like. Continuing with FIGURE 9, each process requesting access 168 is bound to a resource access table (RAT) 164 via respective links 165, 166 and 167. The resource access tables 164 contain entries or statements that may be expressed in a high level programming language. Further coupled as inputs to operating system 162 are the plurality of processes requesting access 168, i.e., applications running on system 160 as indicated by process requesting access 1, process requesting access 2 and so on to process requesting access N. In the description to follow, the words application and process will be used interchangeably, referring generally to an application program as distinguished from an operating system. Such application programs are provided to accomplish specific operations such as spread sheets or word processing or communications and the like. In general, each of these processes or applications 168 will, during their operation, require access and use of various ones of the resources 172 coupled to the operating system 162 of the computer system 160 of the present disclosure. Coupled as inputs to the various processes 168 or applications 168 is provision for entering a user identifier and/or password for each respective process 168 or application 168 that will be requesting access to a resource as will be described further hereinbelow. The entry of the user identifier and/or password is represented by the functional block 170 user ID and password.

[0057] Referring further to FIGURE 9, an operating system 162 for computer system 160 is illustrated, such as a Microsoft Windows based system, a UNIX system, a DOS system or the like. An operating system is the primary method by which a computer interfaces with the various resources including, for example, the peripheral systems such as memory storage devices, printers, modems and a process or application running on said computer. The operating system 162 provides the user with a platform on which to run programs, i.e., applications or processes 168 wherein the user can access the program and have the program access various peripherals through its interaction

with the operating system. Operating system 162 therefore provides the necessary commands to access various resources 172 on the system. Further, the operating system 162 is restricted to allocate only those resources defined in the resource access tables 164, which define resources 172 to be associated with a given process 168, based upon the needs of the process 168.

[0058] The operating system 162 (OS 162) receives as inputs an indication of which of the process blocks 168 is running. In one embodiment the OS 162 may include or be responsive to a System Administrator function to set the parameters of the access control for the resources 172. The OS 162 in conjunction with the resource access table 164 then selects which of the resources 172 is authorized access. It is important to note that it is a request from a process 168 during the running of that process 168 that is utilized by the OS 162 to grant access to the resource access table 164.

[0059] Continuing with FIGURE 9, the functional block 170 indicates both a user of the system and information that may be provided by the user to gain access to the system or its resources. The user of the system can access any given resource appropriate to the application that is provided in the dedicated system. In general, a process or an application is an executable file which may be referred to as "*.EXE", the "*" defining a wild card name of one or more characters representing an executable file or program. For example, one well known word processing program has an executable file name of WP.EXE. The user can enter the term "WP" and "launch" that program. The program will then run in a conventional manner.

[0060] In operation of the system 160 of FIGURE 9, by way of example, if a word processing program were being operated and, on the same computer, a user had the ability to operate an accounting program, the word processor would be provided access to certain regions on a disk and the files associated therewith. The user could retrieve these files, delete them, modify them and restore them. However, the user would not be allowed through the word processor to access the accounting database for any purpose,

since operation of the word processor process does not require this. In another example, if a modem were provided, this would not usually be a resource that was available to a word processor. The modem, for example, could only be accessed by a communications program.

[0061] In an example of the operation of a process based security system, reference is made to Table 2:

TABLE 2

Step	Process or application name	Resource name	Rights Mask
1	C:*.*/S	C:*.*/S E:\LOGIN\LOGIN.EXE	Full access Execute only
2	E:\LOGIN\LOGIN.EXE	E:\SYSTEM\PASSWORDS E:\PROGRAMS\MENU\MENU.EXE	Read only Execute only
3	E;\PROGRAMS\MENU\MENU.EXE	E:\PROGRAMS\MENU\SCREENS E:\PROGRAMS*.EXE/S E:\PROGRAMS*.COM/S	Read only Execute only Execute only
4	E:\PROGRAMS\WP51\WP.EXE	F:\LIBRARY*.* G:\COMMON*.*/S	Full access Full access

For purposes of illustration, the example in Table 2 applies to a personal computer (PC) which is attached to a network and running an MS DOS operating system that is provided with process-based security. Although a PC is usually considered a general purpose system, the simplicity of the illustration provided by Table 1 applies equally well to a dedicated computer system. When the computer is started, as in step 1 in Table 1 described previously, any process to be run with the C:\ drive (denoted with the wild card designation *.*) and its sub-directories (denoted with the /S option on the end of the process name) is provided full access to any resource on the C:\ drive. Note also that the user can execute the resource E:\LOGIN\LOGIN.EXE from the network but

that all other resources from the network are not available to the computer at this time as being limited by the statement E:\LOGIN\LOGIN.EXE. This statement will be described further in the next paragraph. This example, so far, represents a user, on a computer, who can log into a network, but has not done so. In essence, the user can do anything with his or her local resources, but nothing with network resources, until they are identified to the network with the login program.

[0062] In step 2 in Table 2, when the user executes the E:\LOGIN\LOGIN.EXE process, the process changes from something on C:\ to LOGIN.EXE which is permitted to read the E:\SYSTEM\PASSWORDS file and execute the E:\PROGRAMS\MENU\MENU.EXE program. The file LOGIN.EXE is the network's method of identifying users of the network. Execution of LOGIN.EXE will verify the user through its read-only access to the E:\SYSTEM\PASSWORDS file. If the user is verified as a valid user, LOGIN.EXE will pass control on to step 3 and the process E:\PROGRAMS\MENU\MENU.EXE.

[0063] In step 3, when the file MENU.EXE is executed, it will read the appropriate menu options from its SCREENS file and display it for the user. MENU.EXE controls what programs can be executed and as such, it has been given rights to execute any program in the E:\PROGRAMS directory or any of E:\PROGRAMS sub-directories (this is denoted with the /S option after the partial wild card name *.EXE and *.COM as listed in the resources column of Table 2). In step 4, in the event the user executes the WP.EXE program, this process has full access to a local F:\LIBRARY directory, a shared G:\COMMON directory and the sub-directories of G:\COMMON. Step 4 may also represent a network, where personal files are stored in a user-related directory (F:\LIBRARY) and company shared documents are stored in a common directory (G:\COMMON).

[0064] In the preceding examples illustrated by Table 2, it can be seen that the user, because of the table which must be accessed during a resource request, cannot obtain

access to the PASSWORDS file by any other process except via the LOGIN.EXE process. This process also determines how the user can deal with that particular file.

[0065] Referring now to FIGURE 10 there is illustrated a flowchart of the operation of the illustrative embodiment of the process-based security system of FIGURE 9. The flow begins with the Start block 200 and proceeds to block 204 to load the application program. From block 204 the flow proceeds to block 206 to start the application which is followed by decision block 208 to determine whether a resource is requested by the application. If the result of the determination is negative then the flow follows the N path back to the entry to the Start Application block 206. If the determination in block 208 is affirmative then the flow proceeds to block 218 wherein a step to read the respective resource access table 164 for the requesting application 168 is performed.

[0066] Continuing further with FIGURE 10, upon reading the resource access table 164 for the requested application in block 218 the flow proceeds to block 220 wherein the system 160 interprets the entries or statements in the resource access table 166 to identify the commands of the execution path and the sequence of operations contained in it. The flow thereupon proceeds to decision block 222 wherein a determination is made as to whether the request for resource access matches the application in operation. If the determination is negative, then the flow proceeds to block 216 wherein access is denied and thereupon is routed back to the start application block 206. If, however, the determination in decision block 222 is affirmative, then the flow proceeds along the Y path to block 226 wherein access is granted to the requested resource and the flow returns to the main program to execute the application or process as indicated at block 234. It will be appreciated that the security access is provided by the reading and interpreting of the resource access table 164 entries or statements which specify the resources needed for the particular application or process and the execution path for access to those resources. Thus, access to resources is limited to only those resources that are needed and requested by the particular application or process that is in operation.

[0067] A study of FIGUREs 9 and 10 described hereinabove will reveal the following operational characteristics of a process-based security system for dedicated or single-purpose computer systems. Upon launching an application in a process-based system, the access rights are associated or bound to the launched application as indicated by links 165, 166 and 167 in FIGURE 9. During the request for access to the needed resource(s), the access rights associated with that program are checked. In a general purpose system, security checks impede processing by interrupting the OS while the security check is performed each time a user requests a resource. In a dedicated system running one process or a single process combination, only one request for resource access is required; if several process combinations are provided, the system selectively allows access to the resources appropriate to the process requesting access. In either case, the request occurs during the initial steps of the process. Further, the security access is performed by matching the conditions present upon launching the application such as the program identity, user identity and password, execution path through the directory, etc. with the resource access entries or statements in the respective resource access table 164. Once the application is launched, read and write calls are no longer checked, and the resource access table 164 controls "the traffic" - the execution path through the directory. As an example, in a web server application, the steps, briefly, would be: (1) turn on the web server; (2) launch the application; (3) read and interpret the resource access table entries; (4) grant the needed access; and (5) execute the application, including the allowed resources.

[0068] During the interpretation step 220 of FIGURE 10 of the illustrated embodiment, resource access table entries are interpreted, one character at a time, instead of merely reading a resource name associated with a listed process or merely making a string comparison, because of the presence of meta symbols embedded into the entries in the respective resource access table 164. Meta symbols, as disclosed herein, are textual devices which may be inserted into resource access table entries as second-order data or instructions to supply additional related information or modify the

interpretation of the entry in some way. In the comparison process to find a resource in the resource access table, entries do not have to be static. Entries in the resource access table can have meta symbols to allow for context sensitivity to the process making the request. Table 3 presents some examples of meta symbols developed for the embodiment of the present disclosure which may be included in a resource access table entry. Meta symbols are assigned - and construed - in a UNIX environment.

TABLE 3

Meta Symbol	Definition / Meaning
$P	If the requested resource name matches to this point, consider the entry a match (path wild-card).
$C	The particular character in the requested resource name matches this symbol no matter what (character wild-card).
$D	For a single level of depth in the directory, this symbol means a match (directory wild-card).
$$	Requested resource name must match a $ at this point.
$S	Requested resource name must end (suffix) in the text following this symbol.
$U	Requested resource name must have the user name of the user that initiated the process making the request, at this point.
$G	Requested resource name must have the group name of the group that initiated the process making the request, at this group.

As described in the foregoing, to provide process-based security access in a single-purpose "appliance" computer system, a resource access table (RAT) is bound to, i.e., associated with, the requesting process when the process or application is launched. The RAT contains entries in which the defined execution paths, i.e., process paths, are modified using meta symbols. These meta symbols provide instructions for interpreting the process or execution paths. For example, meta symbol entries enable the system to determine which part(s) of an entry in a RAT must be matched character-by-character to produce a valid comparison or which part(s) may be ignored or which part(s) has a substituted instruction, etc., in order to be granted the security access rights associated with that particular entry in the RAT. Each entry or statement in the RAT may correspond to a resource whose access is defined by the entry.

For example, an unmodified entry in a RAT might appear as:

PROGRAMS/WP/WP.EXE

and the resources to be associated therewith might be:

/HOME/$U/$P.

So, when a user initiates a program operation the command string is compared to the RAT entry. In this example, the meta symbol $U means that the current user name is substituted into the entry and permitted access to the respective resource. Similarly, the $P modifies the RAT entry and means that the rest of the path is ignored, i.e., it is "matched" no matter what the rest of the path is.

[0069] Example No. 1: Referring now to FIGURE 11, suppose the user Q is operating in the home directory and wishes to delete a file xyz. In FIGURE 11, the perspective is that of the operating system. The routine begins with the start block 240 and proceeds to function block 242 wherein the operating system (OS) receives a request to run DEL program. Thereafter the OS checks in block 244 whether the current user is allowed a DEL command. If not, the flow follows the N path and returns to enter block 242. If so, the flow proceeds to block 246 to load the DEL command and fetch the corresponding access rights from the resource access table (RAT) 166. In this case the RAT 166 entry is the statement: /HOME/$U/$P which defines access rights in the HOME directory for the $U current user within which access is allowed $P from this point on, i.e., is unrestricted in directory depth per Table 3.

[0070] Continuing with FIGURE 11, in the next step, at decision block 248, the OS determines whether the access rights match the current user and if affirmative, the flow proceeds along the Y path to another decision block 250. There, if both the HOME directory and the current user $U are matched, the routine advances to block 252 where access is granted and the DEL command is allowed to be executed. The routine returns to the main program in block 254. In either case, in blocks 248 or 250, the result is that a match did not occur, access is denied and the routine returns to the entry of block 242.

[0071] Example #2: Suppose the user is operating in the HOME directory, and wishes to run a word processor (e.g. WordPerfect). The word processor program (application or process) is in the directory: /PROGRAMS/WP/WP.EXE. Here, the resource access table statement is: /HOME/$U/$S.WP, where $S is used as an intervening suffix. This statement limits access to files in the user's home directory (/HOME/$U/) that end in the characters .WP ($S.WP).

[0072] It will be appreciated in the foregoing example that any resource can be moved to any place in an execution path it is desired, merely by defining the access rights for that path in the Resource Access Table. Thus, the access rights "move" with the new placement of the resource. Further, many resources, e.g., utility programs, can be wild carded into part of an execution path. In effect, these programs are executed, not out of the original program or process but out of the resource access table 166. This provides a simple way to limit access rights - merely by statements in the resource access table. Moreover, since the substituted directory path identified the word processor WP in its execution path - and not some other resource such as an EXCEL program - access rights to EXCEL (or any other program that may be part of the system) are excluded from the WP program execution path.

[0073] Suppose, alternatively, the user is running EXCEL and wishes to use a spell check resource. Unless that spell check resource, which resides in the WP program, is included in the allowed access rights of the RAT entries for the EXCEL program any user attempt to access it from EXCEL will be denied. It will thus be appreciated that the process-based security described hereinabove provides the advantages of (1) preventing users from loading their own applications on a dedicated system configured according to the present disclosure; and (2) preventing users from attempting to access files via uncontrolled ways such as trying to load in a hex editor, e.g., to obtain access to accounting or other sensitive files.

[0074] Example No. 3: Consider a web server (WS) which can execute any common gateway interface (CGI) serving a plurality of companies, e.g., A, B, C, D, E and F. Prefixing is used to distinguish whose CGI is allowed execution (e.g., ABC/CGI for access to ABC/DATA directory but which may exclude DEF/CGI) by substitution according to a $E(#) meta symbol that identifies the path that is executable out of the original structure in the RAT. In the RAT, as illustrated in Table 4 hereinbelow, it is seen that there are two kinds of entries instead of one: one statement for the web server, another for the CGI for which access rights are defined. Table 4 illustrates a fragment of the RAT for Example 3.

TABLE 4

Web Server	**CGI**	**Rights**
/HTTP		
	/PROGRAMS/$D/$P	Execution
/PROGRAMS/$D/$P		
	$E(2)/DATA/$P	Read, Write

Here, the path /PROGRAMS/ABC is granted access, according to the statement $E(2)/DATA/$P.

[0075] The resource access table 166 entries, thus modified by meta symbols, as described hereinabove, define both the access to resources and the execution path through the directory. The resource access table 166, uniquely determined for the dedicated, single purpose system, is called by the request for access made by the application or process. Thus, the entries in the resource access table are, at the same time, both statements of the access rights and statements of the execution path. In some operations, for example, a meta symbol (identified by a $ followed by a character) inserted into a statement in the resource access table may provide for, referring to Table 3:

(1) association of user identity information with the application or process (user ID and a password, e.g.);

(2) substitution of one user or a group of users for another user;

(3) substitution of a part of one execution path for another one;

(4) specifying at what point in the directory path the access begins, or how far into the directory the access rights extend; and

(5) specifying an access path limited to a particular file name or keyed to the access of a particular file. The meta symbols enable modifications to the entries in the resource access table 166 with instructions that define, on the fly, the particular access rights available to a particular process. Thus, instead of just performing a string comparison of the access rights string (with a predetermined set of access rights) the string is read and interpreted, based on its content, as it proceeds on the execution path.

[0076] In summary, the process-based security as disclosed hereinabove is most efficiently applied to specific functions. The operating system 162 of the dedicated, single-purpose system 160 is bundled only with the specific applications 168 needed including the resource access tables 166 and the necessary code to implement the use of the meta symbols and the process-based security access. Only internal resources are affected. Requests for access to resources 172 are processed from within the particular process or application 168 before invoking the operating system 162 but before the request handler is invoked.

[0077] With reference to FIG 12, a multi-user process based security system is shown. In the case where the computing environment is a computer, the multiplicity of users may be accessing the system sequentially. In the case where the computer environment is shared, such as a server with a multiplicity of clients, the users may be accessing the system simultaneously.

[0078] A first user 300 is authenticated by an authentication agent in the operating system 60 or process-based security module 76 of a computer. Once the first user 300

has established an authenticated identity, the process based security system loads the first user resource access table 312 in database 310. The first user resource access table 312 includes permissions, establishing the resources available to each process that is available to the first user. A first process permission table 314 includes a list of files, directories and other processes that may be accessed by the first user 300 through the first process 302. A second process permission table 316 includes a list of files, directories and other processes that may be accessed by the first user 300 through the second process 306. The first process 302 sends calls to the OS 60, requesting access to a resource 324. A process-based security module 76 consults database 310 for the first user resource access table 312, including the first process resource access table 314. If the requested resource 324 is identified in the first process resource access table 314 of the first user resource access table 312, first process 302 is given access to the requested resource 324.

[0079] When the first user 300 accesses a second process 306, the second process 306 is given access to resources 326 in accordance with a second process resource access table 316 in the first user resource access table 312. The resources 324 available to the first user 300 in the first process 302 will only be available to the first user 300 in the second process 306 where the resource 324 is identified as available to the first and second processes in their respective process resource access table in the first user resource access table 312.

[0080] A second user 304 is authenticated by the operating system 60. The process-based security module 76 reads the second user resource access table 318 in database 310. When the second user 304 accesses the first process 302, the process-based security module 76 checks the first process 302 access permissions with reference to the second process resource access table 320 of the second user resource access table 318. If the second user 304 does not have permission to access the second process 306, there is no second process resource access table in the second user resource access table 318, or the second process resource access table for that user is given a null value. A

third process 308 accessed by the second user 304 is governed by the third process resource access table 322 of the second user resource access table 318.

[0081] The resources available to the first process 302 depend on the identity of the user that has been identified to the system. The use of a user name meta-symbol ($U) in the resource access tables allows the system to identify resources based on the name of the user associated with the process. For example, the resource access table may provide permission for each user to a directory that has been named using the user name. The multi-user process based security system is particularly useful where the system is a web-server or any system having multiple users and a need to control access.

[0082] Because the operating system checks the authorization of every resource call, the security of data, such as password files, does not need to depend on encryption or other forms of masking. Typical prior art systems do not save passwords in cleartext, but instead save hashes of the passwords. When the password is set using a set password function, the password is hashed and stored in association with a username. A login process may request a username and password. The password is hashed and the username is used to retrieve the hash of the password associated with the username. If the two hashes match, the user is authenticated.

[0083] In accordance with the preferred embodiment, a given resource can only be accessed by an authorized user using an authorized process. This allows a password file to be stored as cleartext, which allows greater flexibility in the use of the password in key exchange protocols.

[0084] With reference to FIG 13, a flowchart of an authentication process is shown. The authentication process involves interactions between a user, an authenticating process and an authentication module. The user may be an individual interacting with a single machine, a process accessing an authenticating process, a client in

communication with a server, or any other interactive source of data. The authenticating process may be any process that makes an authentication call. A typical authenticating process is a login process. Any process that requires or needs authentication of a user may be an authenticating process. Some of the functions ascribed to the client or user may be performed by the authenticating process.

[0085] The authentication module works in conjunction with the process based security system to authenticate users to any process that calls on it. In the simplest embodiment, the authentication module receives usernames and passwords and compares the received password with a stored password associated with the username. In accordance with the preferred embodiment, the authentication module uses a handshaking operation to authenticate the user.

[0086] The user or client initiates the authentication process in step 330. The authentication process may be initiated by executing a login program, by executing a task that calls on a login program, or by selecting some function in a program that requires authentication before it will proceed. When the authentication process has been initiated, the authenticating process sends a request for a random number (RN) from the authentication module at step 332.

[0087] When the authenticating process sends a request for a random number to the authentication module, the operating system using process-based security will check to see if the authenticating process is authorized to access the authentication module. Only the processes listed in the resource access table will be able to access the authentication module.

[0088] The authentication module generates a random number (RN) in step 334. In accordance with the preferred embodiment, the random number (RN) is a sixteen byte cryptographically strong random number. In accordance with the preferred embodiment, the system uses random numbers that are cryptographically strong

random numbers, created using processed noise. Other forms of random numbers may be used, as appropriate, including pseudo-random numbers. Pseudo-random numbers may be necessary in protocols where the random numbers need to be reproducible. Preferably, the random numbers are sixteen byte random numbers, although any length random number may be used as appropriate. The authentication module modifies the authenticating process' task structure to reflect the pending authentication request and to restrict access to data storage where the random number (RN) will be stored in step 336. The random number is stored (RNs) at a designated storage location with restricted access in step 338. The random number (RNa) is also sent to the authenticating process in step 340. RNs = RNa

[0089] The user enters a username (USERID) and a password (PWa) in step 342. In a client/server environment, where a process run on the client machine is calling an authenticating process on the server, the username (USERID) and password (PWa) may be kept at the client. In this case, the authenticating process may forward the random number (RNa) to the client. The client uses the password (PWa) and the random number (RNa) to generate a hash H(PWa,RNa) at step 344. In accordance with the preferred embodiment the password (PWa) and the random number (RNa) are concatenated, with the concatenation serving as the data hashed. A keyed hash function, such as a keyed MD5 hash, may use the password (PWa) as the data hashed and the random number (RNa) as the key (K).

[0090] In a local environment where a user is communicating directly with the authenticating process, the user submits the username (USERID) and password (PWa) to the authenticating process. The authenticating process generates the hash H(PWa,RNa).

[0091] In either case, the username (USERID) and the hash H(PWa,RNa) is sent to the authentication module in step 346. The authentication module checks the task structure of the authenticating process to determine if there is an outstanding request for

authentication in step 348. If there is no outstanding request for authentication, the authentication module does not proceed with the authentication process. This deters a malicious user from using a brute force attack against an unchanged stored random number (RNs) by submitting false hashes H(??, RNs) until authentication is achieved. By performing this check of the task structure, the authentication process changes the random number (RNs) for each attempted authentication, reducing the effectiveness of a brute force attack.

[0092] If the task structure of the authenticating process shows a pending request for authentication, the authentication module retrieves the stored random number (RNs) in step 350. The authentication module retrieves a stored password (PWs) associated with the username (USERID) at step 352. The authentication module uses the stored password (PWs) and the stored random number (RNs) to calculate a hash H(PWs,RNs) at step 354. The received hash H(PWa,RNa) is compared to the calculated hash H(PWs,RNs) at step 356. If the hashes are equal, H(PWa,RNa) = H(PWs,RNs), then the user is authenticated. If the hashes are not equal, the authentication process fails.

[0093] In either instance, the authentication module modifies the task structure of the authenticating process to reflect the completion of the pending authentication process at step 358. This prevents further attempts at authentication without generating a new random number (RN). The authentication module may set the user as the username (USERID) in the task structure of the authenticating process in step 360. The authentication is communicated to the authenticating process, which may set the user as the username (USERID) for the application settings in step 362.

[0094] With reference to FIG 14, a flowchart for a set password routine is shown. A set password process communicates with an authentication module. The set password process receives an entered username (USERID) in step 635. The users present password (PW) is entered in step 368. Both the username (USERID) and the password (PW) are sent to the authentication module, where an authentication process is

performed in step 366. If the password (PW) is associated with the username (USERID), the authentication is conformed in step 369 and the set password program requests a new password (PWN) in step 370. The new password (PWN) is sent to the authentication module. The authentication module sets the password (PW) equal to the new password (PWN) and saves the associated username (USERID) and password (PW) in step 372. If the user cannot be authenticated, the process stops.

[0095] With reference to FIG 15, a flowchart for a key exchange process is shown. The key exchange protocol is shown as conducted between a client and server, where the server is operating with a process-based security system. Those having skill in the art will recognize that the key exchange can be performed between any process and a process-based security operating system. In accordance with the preferred embodiment, the functions ascribed to the server are performed primarily by an authentication module operating in conjunction with the operating system.

[0096] The client, or more specifically a process operating in a client relationship with a server, requests a key exchange from the server in step 374. The client sends the username (USERID) to the server in step 375, unless the user has already been authenticated to the system, in which case the server uses the authenticated username. A user enters a password (PW) at the client at step 377. The password (PW) is typically not transmitted to the server.

[0097] The authentication module in step 376 modifies the client task structure to reflect the key exchange process initiation. The server generates a first random number (RN1) in step 378 and sends the first random number (RN1) to the client. In accordance with the preferred embodiment, the system uses random numbers that are cryptographically strong random numbers, created using processed noise. Other forms of random numbers may be used, as appropriate, including pseudo-random numbers. Pseudo-random numbers may be necessary in protocols where the random numbers

need to be reproducible. Preferably, the random numbers are sixteen byte random numbers, although any length random number may be used as appropriate. Depending on the hash function used, the length of the random number and the strength of the randomization function may affect the strength of the key generated, so the random number generation process used should be chosen accordingly.

[0098] The server retrieves the password (PW) associated with the username (USERID) from data storage in step 382. Both the client in step 382 and the server in step 384 independently calculate a hash H(PW,RN1) based on the password (PW) and the first random number (RN1). In accordance with the preferred embodiment, a keyed MD5 signature function is used as the key generating hash function. Other signature or hash functions with sufficient pseudo-random distributions may be used. A first key (K1) is set as equal to the hash H(PW,RN1) in step 386 at the client and step 388 at the server. In one embodiment, the first key (K1) may be used as a symmetric key for all communications between the client and server for the session. In the preferred embodiment, the server sends a second random number (RN2) to the client in step 392, and a second hash is performed by both the client in step 394 and server at step 396 H(PW,RN2) to generate a second key (K2) at step 400 for the client and 398 for the server. The first key (K1) may be used to symmetrically encrypt communications from the server to the client, while the second key (K2) may be used to symmetrically encrypt communications from the client to the server. Once the keys are generated, the authentication module modify the task structure for the client in step 402, ending the key generation process.

[0099] The process-based security system can be used to facilitate secure financial transactions over a network. A typical Internet commerce system allows users to make purchases using a credit card. In order to save the user time and encourage further purchases, the Internet commerce system may save the credit card number, as well as other data used to validate the credit card number such as the expiration date or CCV number. In the event that the Internet commerce system server is compromised, either

by hackers or insiders, the credit card numbers may be stolen and abused. Implementing a process-based security system on the Internet commerce system server could secure the credit card number database, but in some cases the implementation may be too extensive a change.

[0100] With reference to FIG 16, a flowchart for a secured financial transaction in accordance with one embodiment is shown. The transaction is performed by a buyer at a point-of-sale server (POS). The point-of-sale may be a POS terminal at a retail store or a personal computer communicating with an Internet commerce server or any other server operative in creating a financial transaction between a buyer and a financial institution. The point-of-sale server is communicably connected to a process-based security server and a financial server.

[0101] When the buyer initiates the purchase at the point-of-sale server at step 404, the buyer is typically authenticated to the point-of-sale server using a standard authentication technique. With knowledge of the buyer's identity, the point-of-sale server retrieves one or more stored portions of credit card numbers in step 406, allowing the buyer to choose one of those credit card to complete the transaction. Because the storage of credit card numbers at the point-of-sale server is insecure, in accordance with one embodiment, the point-of-sale server displays only the last four numbers of the credit card numbers, uniquely identifying the buyer's credit cards without revealing the actual credit card number.

[0102] When the buyer has selected a credit-card to use for the purchase in step 408, the point-of-sale server correlates a credit card identifier with the portion of the credit card number that has been selected. The credit card identifier is a number previously defined by the process-based security server to serve as a representation of the credit card in communication between the point-of-sale server and the process-based security server. The point-of-sale server sends the credit card identifier to the process-based

security server, along with any necessary transaction data such as the amount of purchase in step 410.

[0103] The process-based security server receives the credit card identifier and retrieves the associated credit card number, expiration data and CCV number from secured storage in step 412. Because the process-based security server can control access to the sensitive data, the credit card numbers stored at the process-based security server cannot be compromised by hackers or malicious insiders. The credit card number, along with the expiration date and the CCV number, are sent to a financial server associated with the chosen credit card in step 414. The financial server processes the transaction and sends a message either approving or denying the transaction to the point-of-sale server in step 416. In some embodiments, the message may be sent to the process-based security server to be forwarded to the point-of-sale server. If the transaction has been approved, the point-of-sale server completes the transaction with the buyer in step 418. If the transaction has been denied, the point-of-sale server informs the buyer of the denial. In either case, the process-based security server may generate a new credit card identifier in step 420. The new credit card identifier is stored in the process-based server in association with the credit card number data and sent to the point-of-sale server to replace the old credit card identifier in step 422. This continual refreshing of the credit card identifier limits any possible damage caused by intercepting a communication containing the credit card identifier, as the identifier ceases to be valid after one attempted use.

[0104] Another use of the process-based security system is for secure file transfer. With reference to FIG 17, a flowchart for a secure file transfer process is shown. The process allows Client A, connected to a process based security server, to securely transfer a file to Client B. Client A initiates the session with the process-based security server in step 424. The process based security server authenticates Client A, using the authentication protocol outlined previously in step 426. Client A then transmits a file in step 428 to the process-based security server for storage in a location that is

accessible to Client A and Client B in step 430. When Client B initiates a session in step 432, the process-based security server authenticates the identity of Client B in step 434. Once authenticated, Client B is then permitted to access the file stored in the location accessible to Client A and Client B in step 436. Transmission of the file between the clients and the process-based security server is typically encrypted, using SSL or some other encryption protocol to secure the data during the network transmissions.

[0105] With reference to FIG 18, a flowchart of a boot sequence in accordance with one embodiment is shown. When the boot sequence is initiated in function block 438, the process-based security server checks the hard drive of the process-based security server to check for a boot sector in function block 440. Normally, the boot sector will be present on the hard drive, but in the case where a new hard drive has been installed or the boot sector of the hard drive has been damaged, there will not be a working boot sector. In the case where the hard disk is damaged but the hard disk still has a working boot sector, a function may be present to allow a user to force the device to boot off of the flash memory. If there is a boot sector detected in decision block 442, the process follows the YES path to boot the process-based security server from the hard drive boot sector in function block. If no boot sector is detected, the process follows the NO path to function block 446 and the process-based security server boots from a flash memory. The flash memory boot causes the hard drive to be formatted in function block 448 and populates the hard drive with the process-based security software in function block 450.

[0106] A process-based security system may be used on a server in a network computing environment, on any computer or computing device, either in isolation or working as a client connected to a server. Other digital devices suitable for implementing operating system level access protection, such as laptops, hand-held computing devices, personal digital assistants (PDA), or cellular telephones, may implement process-based security.

[0107] Referring to Figure 19, a flowchart of a table building process in accordance with one embodiment is shown. In creating a resource access table for use in a system, or for a user of a system, each process available to the system or user may be analyzed. A process may be a program, driver, applet, script, executable image or basically any form of instruction that could access a resource on a system. The implementation of the process will be referred to as code. In function block 500, the table-building process analyzes the code. The process continues to function block 502 where the process identifies resource calls. The code may be analyzed by visual inspection of a written embodiment of the code, by a person capable of recognizing resource calls. The code may be analyzed by running the program in trace mode, such that each resource call of the code can be identified. The code may be analyzed by software designed to identify resource calls and performed automatically or semi-automatically.

[0108] When the resource calls have been identified in function block 502, the identified resource calls are sorted and may be assigned a reference label in function block 504. The resources called by the resource calls are identified at function block 506.

[0109] At function block 508, permissions are assigned to each possible resource called by the code. In accordance with the preferred embodiment, the process resource access table is written in an open form, such that only resources listed in the process resource access table may be accessed. The process resource access table may also be written with a different default, for example, such that any resource may be accessed unless listed in the process resource access table. Using the preferred, open form, Table 1 shows two resources available to E:\LOGIN\LOGIN.EXE, the file E:\SYSTEM\PASSWORDS and the executable E:\PROGRAMS/MENU/MENU.EXE.

[0110] When the resources are assigned permissions, the table-building process writes the process resource access table in the form Process/resource/permissions in function block 510. If there is no access to a resource, in an open form, no entry is

added to the resource access table. Decision block 520 determines if there are further resources requiring permission assignment.

[0111] If further resources remain, the process follows the NO path to function block 508, where the next permission is assigned. If no further resources remain, the process follows the YES path to function block 522 where the process resource table is compiled with other process resource tables into a system or user resource access table. When the resource access table has been compiled, it is saved in a predesignated memory space at function block 524. The resource access table may be saved in the predesignated memory space to protect the resource access table as a resource and allow the resource access process to access the resource access table for use.

[0112] With reference to Figure 20, a process for table-building in accordance with another embodiment. In creating a resource access table for use in a system, or for a user of a system, each process available to the system or user may be analyzed. A process may be a program, driver, applet, script, executable image or basically any form of instruction that could access a resource on a system. The implementation of the process will be referred to as code. In function block 526, the table-building process analyzes the code. The process continues to function block 528 where the process identifies resource calls. The code may be analyzed by visual inspection of a written embodiment of the code, by a person capable of recognizing resource calls. The code may be analyzed by running the program in trace mode, such that each resource call of the code can be identified. The code may be analyzed by software designed to identify resource calls and performed automatically or semi-automatically.

[0113] When the resource calls have been identified in function block 528, the identified resource calls are sorted and may be assigned a reference label in function block 530.

[0114] At function block 532, permissions are assigned to each possible resource called by each resource call. In accordance with the preferred embodiment, the process resource access tables and the resource access table are written in an open form, such that only resources listed in the process resource access table may be accessed. The process resource access table may also be written with a different default, for example, such that any resource may be accessed unless listed in the process resource access table. Using the preferred, open form, Table 1 shows two resources available to E:\LOGIN\LOGIN.EXE, the file E:\SYSTEM\PASSWORDS and the executable E:\PROGRAMS/MENU/MENU.EXE. All other resources are unavailable to the LOGIN.EXE process.

[0115] When the resources are assigned permissions, the table-building process writes the process resource access table in the form Process/resource call/resource/permissions in function block 534. If there is no access to a resource, in an open form, no entry is added to the resource access table. Decision block 536 determines if there are further resources requiring permission assignment.

[0116] If further resources remain, the process follows the NO path to function block 532, where the next permission is assigned. If no further resources remain, the process follows the YES path to function block 538 where the process resource table is compiled with other process resource tables into a system or user resource access table. When the resource access table has been compiled, it is saved in a predesignated memory space at function block 540. The resource access table may be saved in the predesignated memory space to protect the resource access table as a resource and allow the resource access process to access the resource access table for use.

[0117] With reference to Figure 21, a process-based security system using intrusion detection for table building is shown. In this embodiment, when a new application 302 is executed, the process-based security 76 uses standard UNIX permissions 542 to determine resource access 324 for the application 302. As security calls are made by

the application 302, the process-based security sends data regarding the security resource calls to intrusion detection software 544. The intrusion detection software 544 determines the nature of the security resource call and determines patterns in the security calls that would exemplify "normal" behavior for the application 302. The intrusion detection software 544 uses the pattern recognitions and other analysis techniques to determine acceptable resource access for the application 302 and writes a resource access table 312. When the application 302 is run subsequently, the process-based security system 76 uses the resource access table 312 to determine permissions.

[0118] With reference to Figure 22, a flowchart of a table-building process using intrusion detection analysis is shown. When an application is executed in function block 546, the process continues to decision step 548, where the process determines if a resource access table has been defined for the application. If the application has an associated resource access table, the process continues along the YES path to function block 550. The process uses the resource access table.

[0119] If a resource access table has not been associated with the application, the process follows the NO path to function block 552, and the UNIX permissions are used. The process then cycles at decision step 554, waiting for a security resource access. When a security resource access occurs, the process continues to function block 556 where access is determined by the UNIX permissions. The process then reports the security resource access to the intrusion detection software at function block 558. The data is analyzed in step 560 and a resource access table is written at function block 562. If sufficient data has been collected at decision block 564, the process defines the resource access table for the application at function block 566. If sufficient data has not been collected, the process returns to decision block 554 to await a subsequent security resource call.

[0120] Although the illustrative embodiment has been described in detail, it should be understood that various changes, substitutions and alterations can be made therein

without departing from the spirit and scope of the invention as defined by the appended claims.